How to Automate Currency Conversion in Your SaaS Billing System

If your SaaS product only bills in USD, you are leaving money on the table. Studies consistently show that customers are more likely to complete a purchase when they see prices in their local currency. For subscription businesses, this translates directly into reduced churn and higher lifetime value.

But multi-currency billing is not just about slapping a currency symbol on a number. You need reliable exchange rates, proper rounding, audit trails, and integration with your payment processor. This guide walks you through building an automated multi-currency billing pipeline from scratch.

Why SaaS Companies Need Multi-Currency Billing

Three forces are pushing SaaS companies toward multi-currency support:

1. **Global expansion is the default.** Even small SaaS products acquire international users from day one. A developer tool launched in San Francisco will have users in London, Berlin, and Tokyo within weeks.
2. **Local currency pricing reduces friction.** When a customer in Japan sees "$49/mo," they have to mentally convert to yen, factor in their bank's exchange markup, and wonder what the final charge will be. Showing "7,350 JPY/mo" removes that friction entirely.
3. **Churn decreases with predictable billing.** Customers billed in foreign currencies see fluctuating charges each month. A $49 subscription might be 42 EUR one month and 45 EUR the next. That unpredictability causes support tickets and cancellations.

The Multi-Currency Billing Architecture

Here is the flow for a typical SaaS billing system with multi-currency support:

1. **User signs up** -- The customer creates an account and enters their billing details.

2. **Detect currency** -- Infer the preferred currency from the customer's country, IP address, or explicit selection.
3. **Store preference** -- Save the customer's currency preference in your database alongside their profile.
4. **Invoice generation** -- At billing time, generate an invoice in the customer's preferred currency.
5. **Rate lookup** -- Fetch the current exchange rate from your API (e.g., AllRatesToday).
6. **Conversion** -- Convert the base price to the target currency and apply rounding rules.
7. **Stripe charge** -- Pass the converted amount and currency code to Stripe (or your payment processor).
8. **Store the rate** -- Record the exchange rate used for this transaction in your database for auditing.

Each of these steps can be automated. Let's walk through the implementation.

Step 1: Store Prices in a Base Currency

Pick a single base currency for your internal pricing. USD is the most common choice, but EUR or GBP work too. The key rule: **never store prices in multiple currencies.** You maintain one canonical price and convert on the fly.

Your pricing table should look like this:

```
CREATE TABLE plans (
  id UUID PRIMARY KEY,
  name VARCHAR(100) NOT NULL,
  price_base_cents INTEGER NOT NULL,  -- always in USD cents
  base_currency CHAR(3) DEFAULT 'USD',
  billing_interval VARCHAR(20) NOT NULL  -- 'monthly' or 'yearly'
);
```

Storing prices in cents (or the smallest currency unit) avoids floating-point issues. A $49 plan is stored as `4900`.

Step 2: Fetch Rates at Invoice Time

When it is time to generate an invoice, fetch the current exchange rate. The AllRatesToday API provides real-time rates sourced from Reuters/Refinitiv, which is important for billing accuracy.

```
import AllRatesToday from '@allratestoday/sdk';

const client = new AllRatesToday({ apiKey: process.env.ALLRATESTODAY_API_KEY });

async function convertForInvoice(baseAmountCents, baseCurrency, targetCurrency) {
  // Fetch the current rate
  const response = await client.rates.get({
    base: baseCurrency,
    symbols: [targetCurrency],
  });

  const rate = response.rates[targetCurrency];

  // Convert and round to the smallest currency unit
  const convertedCents = Math.round(baseAmountCents * rate);

  return {
    originalAmountCents: baseAmountCents,
    convertedAmountCents: convertedCents,
    exchangeRate: rate,
    baseCurrency,
    targetCurrency,
    rateTimestamp: response.timestamp,
  };
}

// Example: Convert $49.00 (4900 cents) to EUR
const result = await convertForInvoice(4900, 'USD', 'EUR');
// { convertedAmountCents: 4508, exchangeRate: 0.92, ... }
```

Important: Always fetch rates at invoice generation time, not at page load or sign-up time. Rates change, and you want the most current rate when you actually charge the customer.

Step 3: Lock and Store the Conversion Rate

Once you fetch a rate and generate an invoice, lock that rate. If the customer disputes the charge or you need to issue a refund, you need to know exactly what rate was applied.

```
CREATE TABLE invoices (
  id UUID PRIMARY KEY,
  customer_id UUID NOT NULL REFERENCES customers(id),
  plan_id UUID NOT NULL REFERENCES plans(id),
  base_amount_cents INTEGER NOT NULL,
  base_currency CHAR(3) NOT NULL,
  converted_amount_cents INTEGER NOT NULL,
  target_currency CHAR(3) NOT NULL,
  exchange_rate DECIMAL(12, 6) NOT NULL,
  rate_timestamp TIMESTAMPTZ NOT NULL,
  rate_source VARCHAR(50) DEFAULT 'allratestoday',
  status VARCHAR(20) DEFAULT 'pending',
  created_at TIMESTAMPTZ DEFAULT NOW()
);
```

Key columns: - `exchange_rate` -- The exact rate used for this invoice. - `rate_timestamp` -- When the rate was fetched, not when the invoice was created. - `rate_source` -- Which API provided the rate. Useful for audit and compliance.

Step 4: Integrate with Stripe for Local Currency Charging

Stripe supports charging in 135+ currencies. Once you have the converted amount, pass it directly to the Stripe API with the appropriate currency code.

```
import Stripe from 'stripe';

const stripe = new Stripe(process.env.STRIPE_SECRET_KEY);

async function chargeCustomer(customer, invoice) {
  const paymentIntent = await stripe.paymentIntents.create({
    amount: invoice.convertedAmountCents,
    currency: invoice.targetCurrency.toLowerCase(),
    customer: customer.stripeCustomerId,
    payment_method: customer.defaultPaymentMethodId,
    off_session: true,
    confirm: true,
    metadata: {
      invoice_id: invoice.id,
      exchange_rate: invoice.exchangeRate.toString(),
      base_amount_cents: invoice.baseAmountCents.toString(),
      base_currency: invoice.baseCurrency,
    },
  });
```

```
  return paymentIntent;
}
```

Notice how we store the exchange rate and base amount in Stripe's metadata. This creates a secondary audit trail inside your payment processor.

Handling Edge Cases

Rate Fluctuations Between Billing Cycles

Exchange rates can shift significantly between monthly billing cycles. A customer paying 45 EUR one month might see 47 EUR the next. Best practices:

- **Set a fluctuation threshold.** If the converted price changes by more than 5% from the previous invoice, notify the customer before charging.
- **Offer rate-lock periods.** Some SaaS companies lock the exchange rate for 3 or 6 months, absorbing small fluctuations.
- **Show the base price.** Always display "USD 49.00 (approximately EUR 45.08 at current rates)" so customers understand the conversion.

Refunds

Always refund in the original charge currency at the original exchange rate. If you charged 4,508 EUR cents, refund 4,508 EUR cents -- do not re-convert from USD at today's rate.

```
async function refundInvoice(invoice) {
  // Refund the exact converted amount, not a re-converted amount
  const refund = await stripe.refunds.create({
    payment_intent: invoice.stripePaymentIntentId,
    amount: invoice.convertedAmountCents,  // original converted amount
  });

  return refund;
}
```

Credit Notes and Partial Refunds

For partial refunds, calculate the partial amount using the original exchange rate:

```
function calculatePartialRefund(invoice, refundPercentage) {
  const baseRefundCents = Math.round(invoice.baseAmountCents * refundPercentage);
```

```
  const convertedRefundCents = Math.round(baseRefundCents * invoice.exchangeRate);

  return {
    baseRefundCents,
    convertedRefundCents,
    exchangeRate: invoice.exchangeRate,  // use the ORIGINAL rate
  };
}
```

Audit Trail and Compliance

Financial regulators in many jurisdictions require you to document the exchange rates used in cross-currency transactions. Your audit trail should capture:

- **The rate source** -- Which API or data provider supplied the rate (e.g., AllRatesToday, sourced from Reuters/Refinitiv).
- **The exact timestamp** -- When the rate was fetched, down to the second.
- **The rate applied** -- The precise rate used for conversion.
- **Both amounts** -- The base currency amount and the converted amount.

AllRatesToday provides institutional-grade rates with timestamps in every response, making compliance straightforward. Each API response includes the data source and the exact time the rate was published.

For EU-based SaaS companies, note that PSD2 and the European Payments Directive have specific requirements around exchange rate disclosure. Always show the customer the exchange rate before charging.

Putting It All Together

Here is a complete billing cycle function that ties all the steps together:

```
async function processBillingCycle(customerId) {
  const customer = await db.customers.findById(customerId);
  const plan = await db.plans.findById(customer.planId);

  // Step 1: Base price is already in USD cents in the plans table

  // Step 2: Fetch the current rate
  const conversion = await convertForInvoice(
    plan.priceBaseCents,
    plan.baseCurrency,
```

```
    customer.preferredCurrency
  );

  // Step 3: Create and store the invoice with the locked rate
  const invoice = await db.invoices.create({
    customerId: customer.id,
    planId: plan.id,
    baseAmountCents: conversion.originalAmountCents,
    baseCurrency: conversion.baseCurrency,
    convertedAmountCents: conversion.convertedAmountCents,
    targetCurrency: conversion.targetCurrency,
    exchangeRate: conversion.exchangeRate,
    rateTimestamp: conversion.rateTimestamp,
    rateSource: 'allratestoday',
  });

  // Step 4: Charge via Stripe in the local currency
  const payment = await chargeCustomer(customer, invoice);

  // Update invoice status
  await db.invoices.update(invoice.id, {
    status: 'paid',
    stripePaymentIntentId: payment.id,
  });

  return { invoice, payment };
}
```

Get Started

Multi-currency billing does not have to be complicated. With a base-currency pricing model, a reliable exchange rate API, and proper rate locking, you can support customers in any currency without building a complex financial system.

AllRatesToday provides the exchange rate data you need: real-time rates from Reuters/Refinitiv, official SDKs for Node.js, Python, and PHP, and a generous free tier to get started. Sign up at allratestoday.com and have multi-currency billing running in an afternoon.